HW06WP_ML

August 6, 2023

1 Classification with the *sklearn* ML Framework

Table of Contents

1.1 Preliminaries

Import datetime

[ ]:
```
from datetime import datetime as dt
now = dt.now()
print('Analysis on', now.strftime("%Y-%m-%d"), 'at', now.strftime("%H:%M %p"))
```

```
Analysis on 2023-08-03 at 14:53 PM
```

Establish current working directory.

[ ]:
```
import os
os.getcwd()
```

[ ]:
```
'/Users/chasecarlson/Documents/GSCM Course Materials/GSCM 575 Machine Learning
in Business/Python Pjojects/GSCM-575-ML/code'
```

Import libraries.

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

A classic application of supervised machine learning classification is customer churn. The ability to successfully forecast a customer of a company's services and products about to no longer be a customer allows the company to commit resources to attempt to salvage the relationship.

The following data file contains information on over 7000 customers of a telecom service, including former customers who left the service plan within the last 30 days the data was collected.

Data: http://web.pdx.edu/~gerbing/data/churn_clean.csv

The data has been cleaned according to the analysis for last week, so no need to repeat the cleaning process, or the data exploration.

1.2 Get and Structure Data

a. Read the cleaned data into a data frame, and display its dimensions.

[ ]:
```
df = pd.read_csv('http://web.pdx.edu/~gerbing/data/churn_clean.csv')
df.shape
```

[ ]:
```
(7032, 10)
```

The data frame has 7,032 rows and 10 columns.

b. Display the variable names and the first six rows of data.

[ ]:
```
df.head(6)
```

[ ]:
```
   Charges  TotalCharges  MtoM  Paperless  Check  Phone  tenure  Dependents  \
0    29.85         29.85     1          1      0      0       1           0
1    56.95       1889.50     0          0      1      1      34           0
2    53.85        108.15     1          1      1      1       2           0
3    42.30       1840.75     0          0      0      0      45           0
4    70.70        151.65     1          1      0      1       2           0
5    99.65        820.50     1          1      0      1       8           0

   Internet  Churn
0         0      0
1         0      0
2         0      1
3         0      0
4         0      1
5         0      1
```

Rename 'tenure' column with capital 'T' to be consistent with other columns:

[ ]:
```
df.rename(columns={'tenure': 'Tenure'}, inplace=True)
df.head()
```

[ ]:
```
   Charges  TotalCharges  MtoM  Paperless  Check  Phone  Tenure  Dependents  \
0    29.85         29.85     1          1      0      0       1           0
1    56.95       1889.50     0          0      1      1      34           0
2    53.85        108.15     1          1      1      1       2           0
3    42.30       1840.75     0          0      0      0      45           0
4    70.70        151.65     1          1      0      1       2           0

   Internet  Churn
0         0      0
```

```
1          0      0
2          0      1
3          0      0
4          0      1
```

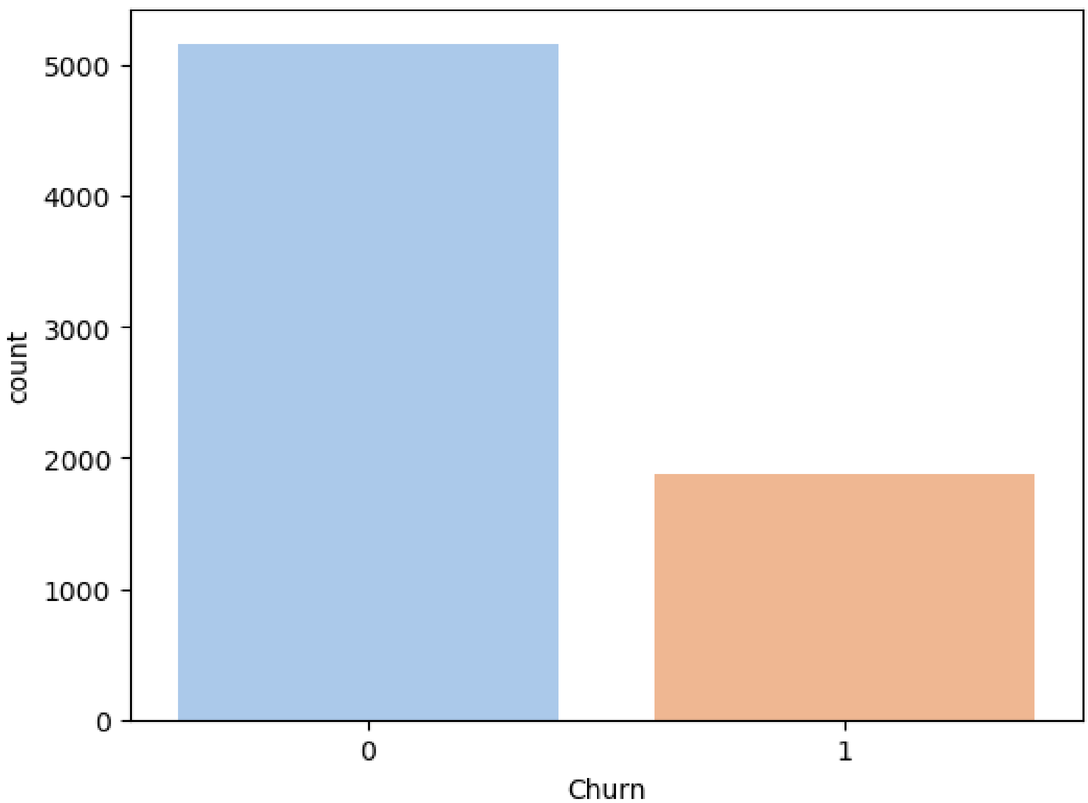
c. Score 1 for Churn. Create the lists of the names of classes *and* features *for the graph later on, then create the data structures from these names.*

[ ]:
```
classes = ['Churn_no', 'Churn_yes']
features = ['Charges', 'TotalCharges', 'MtoM', 'Paperless', 'Check',
            'Phone', 'Tenure', 'Dependents', 'Internet']
X = df[features]
y = df['Churn']
```

Confirm the 1s are Churn and the 0s are not churn using countplot() to view the distribution. Churn customers should be much smaller than non-churn:

[ ]:
```
sns.countplot(df, x = 'Churn', palette='pastel')
```

[ ]: `<Axes: xlabel='Churn', ylabel='count'>`



d. Do a MinMax transformation to get all data into a 0 to 1 range. Verify.

[ ]:
```
from sklearn.preprocessing import MinMaxScaler
mm_scaler = MinMaxScaler()
```

[ ]:
```
scaled_cols = ['Charges', 'TotalCharges', 'Tenure']
df[scaled_cols] = mm_scaler.fit_transform(df[scaled_cols])
df.head()
```

[ ]:
```
    Charges  TotalCharges  MtoM  Paperless  Check  Phone    Tenure  \
0  0.115423      0.001275     1          1      0      0  0.000000
1  0.385075      0.215867     0          0      1      1  0.464789
2  0.354229      0.010310     1          1      1      1  0.014085
3  0.239303      0.210241     0          0      0      0  0.619718
4  0.521891      0.015330     1          1      0      1  0.014085

   Dependents  Internet  Churn
0           0         0      0
1           0         0      0
2           0         0      1
3           0         0      0
4           0         0      1
```

Confirm MinMax transformation:

[ ]:
```
print("Min Values:")
print(df[['Charges', 'TotalCharges', 'Tenure']].min(), "\n")
print("Max Values:")
print(df[['Charges', 'TotalCharges', 'Tenure']].max())
```

```
Min Values:
Charges         0.0
TotalCharges    0.0
Tenure          0.0
dtype: float64

Max Values:
Charges         1.0
TotalCharges    1.0
Tenure          1.0
dtype: float64
```

Reestablish feature and target data structures post-MinMax transformation:

[ ]:
```
classes = ['Churn_no', 'Churn_yes']
features = ['Charges', 'TotalCharges', 'MtoM', 'Paperless', 'Check',
            'Phone', 'Tenure', 'Dependents', 'Internet']
X = df[features]
y = df['Churn']
```

1.3 Grid search: Hyperparameter tuning with cross-validation

e. Do a grid search with a 3-fold cross-validation. Search on the following parameters and values: maximum depth with values of 3 and 4, and maximum features with values of 4, 6, and 8.

Access the decision tree solution algorithm and instantiate with a maximum depth of 4 using DecisionTreeClassifier.

[ ]:
```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(max_depth=4)
dt_model
```

[ ]:
```
DecisionTreeClassifier(max_depth=4)
```

Access KFold cross-validation algorithm and GridSearchCV and create the model:

[ ]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
kf = KFold(n_splits=3, shuffle=True, random_state=1)

params = {'max_depth': [3, 4],
          'max_features': [4, 6, 8]}
grid_search = GridSearchCV(dt_model, param_grid=params, cv=kf,
                           scoring=('accuracy', 'recall', 'precision'),
                           refit=False, return_train_score=True)
grid_search.fit(X,y)
```

[ ]:
```
GridSearchCV(cv=KFold(n_splits=3, random_state=1, shuffle=True),
             estimator=DecisionTreeClassifier(max_depth=4),
             param_grid={'max_depth': [3, 4], 'max_features': [4, 6, 8]},
             refit=False, return_train_score=True,
             scoring=('accuracy', 'recall', 'precision'))
```

f. Display all the results of the cross-validation grid search.

[ ]:
```
df_results = pd.DataFrame(grid_search.cv_results_).round(3)
df_results = df_results.drop(['params'], axis='columns')
df_results.transpose()
```

[ ]:

	0	1	2	3	4	5
mean_fit_time	0.002	0.003	0.003	0.002	0.003	0.004
std_fit_time	0.0	0.0	0.0	0.0	0.0	0.0
mean_score_time	0.002	0.002	0.002	0.002	0.002	0.002
std_score_time	0.0	0.0	0.0	0.0	0.0	0.0
param_max_depth	3	3	3	4	4	4
param_max_features	4	6	8	4	6	8
split0_test_accuracy	0.774	0.797	0.797	0.794	0.796	0.796
split1_test_accuracy	0.747	0.779	0.781	0.76	0.779	0.788
split2_test_accuracy	0.771	0.781	0.778	0.771	0.77	0.784
mean_test_accuracy	0.764	0.786	0.785	0.775	0.782	0.789

std_test_accuracy	0.012	0.008	0.008	0.014	0.01	0.005
rank_test_accuracy	6	2	3	5	4	1
split0_train_accuracy	0.765	0.782	0.782	0.795	0.79	0.789
split1_train_accuracy	0.754	0.788	0.79	0.777	0.791	0.794
split2_train_accuracy	0.778	0.789	0.791	0.783	0.783	0.796
mean_train_accuracy	0.766	0.786	0.787	0.785	0.788	0.793
std_train_accuracy	0.01	0.003	0.004	0.007	0.003	0.003
split0_test_recall	0.282	0.406	0.404	0.488	0.519	0.512
split1_test_recall	0.599	0.361	0.374	0.544	0.518	0.491
split2_test_recall	0.386	0.396	0.355	0.264	0.267	0.458
mean_test_recall	0.422	0.387	0.378	0.432	0.435	0.487
std_test_recall	0.132	0.019	0.02	0.121	0.119	0.022
rank_test_recall	4	5	6	3	2	1
split0_train_recall	0.292	0.376	0.373	0.498	0.511	0.504
split1_train_recall	0.631	0.391	0.399	0.554	0.526	0.494
split2_train_recall	0.41	0.403	0.372	0.277	0.28	0.475
mean_train_recall	0.444	0.39	0.381	0.443	0.439	0.491
std_train_recall	0.14	0.011	0.012	0.12	0.113	0.012
split0_test_precision	0.649	0.684	0.685	0.635	0.629	0.632
split1_test_precision	0.52	0.654	0.653	0.549	0.597	0.63
split2_test_precision	0.628	0.663	0.675	0.712	0.705	0.643
mean_test_precision	0.599	0.667	0.671	0.632	0.644	0.635
std_test_precision	0.056	0.013	0.014	0.067	0.045	0.006
rank_test_precision	6	2	1	5	3	4
split0_train_precision	0.639	0.668	0.669	0.656	0.635	0.637
split1_train_precision	0.531	0.673	0.679	0.585	0.626	0.646
split2_train_precision	0.617	0.664	0.688	0.73	0.728	0.653
mean_train_precision	0.596	0.668	0.678	0.657	0.663	0.645
std_train_precision	0.046	0.004	0.008	0.059	0.046	0.007

g. Display the most relevant results, the means.

Reduce the results to just the desired columns and rename for readability:

[ ]:
```
df_summary = df_results[['param_max_depth', 'param_max_features',
'mean_test_accuracy', 'mean_test_recall', 'mean_test_precision',
'mean_train_accuracy','mean_train_recall', 'mean_train_precision']]
df_summary = df_summary.rename(columns= {
'param_max_depth': 'depth',
'param_max_features': 'features',
'mean_test_accuracy': 'test_accuracy',
'mean_test_recall': 'test_recall',
'mean_test_precision': 'test_precision',
'mean_train_accuracy': 'train_accuracy',
'mean_train_recall': 'train_recall',
'mean_train_precision': 'train_precision'})
df_summary
```

```
[ ]:  depth features  test_accuracy  test_recall  test_precision  train_accuracy  \
0     3        4          0.764        0.422           0.599           0.766
1     3        6          0.786        0.387           0.667           0.786
2     3        8          0.785        0.378           0.671           0.787
3     4        4          0.775        0.432           0.632           0.785
4     4        6          0.782        0.435           0.644           0.788
5     4        8          0.789        0.487           0.635           0.793

   train_recall  train_precision
0         0.444            0.596
1         0.390            0.668
2         0.381            0.678
3         0.443            0.657
4         0.439            0.663
5         0.491            0.645
```

Calculate the variance between train_precision and test_precision for all folds:

```
[ ]: df_summary['precision_variance'] = abs(df_summary['train_precision'] -
      df_summary['test_precision'])
     df_summary
```

```
[ ]:  depth features  test_accuracy  test_recall  test_precision  train_accuracy  \
0     3        4          0.764        0.422           0.599           0.766
1     3        6          0.786        0.387           0.667           0.786
2     3        8          0.785        0.378           0.671           0.787
3     4        4          0.775        0.432           0.632           0.785
4     4        6          0.782        0.435           0.644           0.788
5     4        8          0.789        0.487           0.635           0.793

   train_recall  train_precision  precision_variance
0         0.444            0.596               0.003
1         0.390            0.668               0.001
2         0.381            0.678               0.007
3         0.443            0.657               0.025
4         0.439            0.663               0.019
5         0.491            0.645               0.010
```

h. Main management goal is to detect churners before they churn, which means focus on avoiding false positives. So, focus on precision. Why is the model with a depth of 3 and 6 features a good model to choose?

Based on the above information, the model with a depth of 3 and 6 features appears to be a good model to choose because it has one of the highest precision scores with the smallest variance between the training and testing data at 0.667 test_precision and 0.668 train_precision.

i. Given sufficient fit, estimate the model on the full data set as the best estimates are generally obtained with the most data.

[ ]:
```python
dt_model = DecisionTreeClassifier(max_depth=3, max_features=6)
mf = dt_model.fit(X,y)
```

j. Calculate $\hat{y}$.

[ ]:
```python
y_fit = dt_model.predict(X)
y_fit
```

[ ]:
```
array([0, 0, 0, …, 0, 1, 0])
```

Check results in confusion matrix:

[ ]:
```python
from sklearn.metrics import confusion_matrix
pd.DataFrame(confusion_matrix(y, y_fit))
```

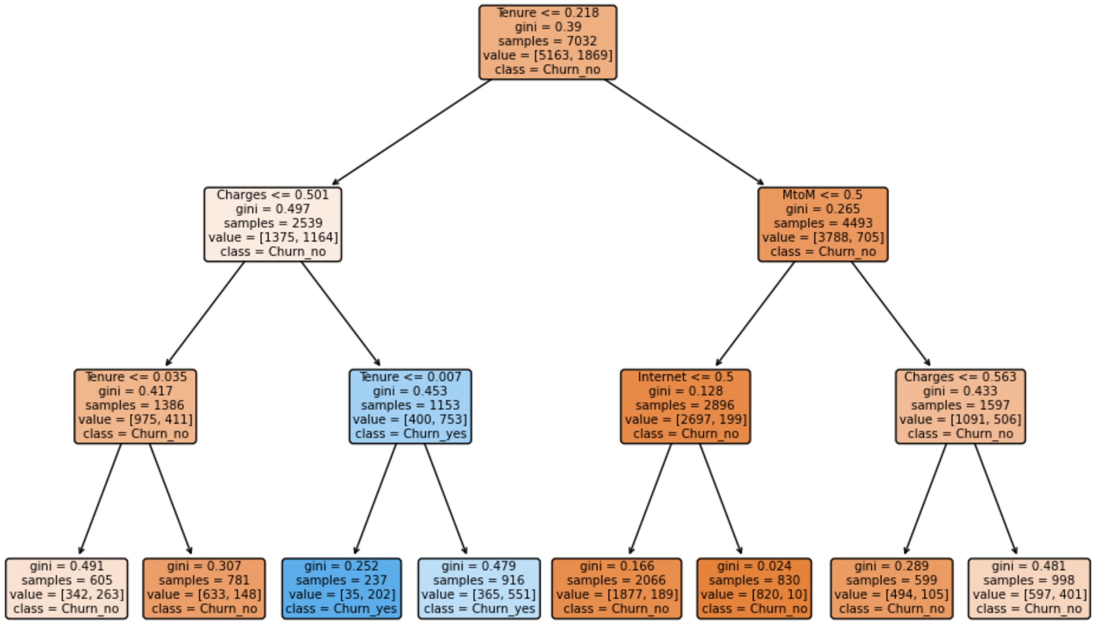
[ ]:

	0	1
0	4763	400
1	1116	753

1.4 Illustrate the Model

k. Draw the tree diagram.

[ ]:
```python
from sklearn import tree
plt.figure(figsize=(12,8))
tree.plot_tree(mf, feature_names=features, class_names=classes,
               rounded=True, filled=True)
plt.savefig('dt_Churn.png')
```



l. Focus on specific leaves.

- *Specify the decision rules that does the best job of detecting churners.* The leaf that does the best job of detecting churners is the leaf third from the left with 202 correctly classified as churn_yes, or 85.23% of total observations.
- *Specify the decision rules that does the worst job in the sense of too many false positives.* The leaf that does the worst job in the sense of too many false positives is the leaf fourth from the left, with 39.85% (365 people) misclassified as positive

1.5 Apply the Model

m. Apply the model to a person who has the following scores.

- Charges = 45
- TotalCharges = 45
- MtoM = 29
- Paperless = 1
- Check = 1
- Phone = 0
- tenure = 0
- Dependents = 1
- Internet = 0

[ ]:
```
X.columns
```

[ ]: Index(['Charges', 'TotalCharges', 'MtoM', 'Paperless', 'Check', 'Phone',
       'Tenure', 'Dependents', 'Internet'],
      dtype='object')

Add the new data and view it in a data frame:

[ ]:
```
X_new = [[45,45,29,1,1,0,0,1,0]]
X_new = pd.DataFrame(X_new)
X_new.columns = X.columns
X_new
```

[ ]:
```
   Charges  TotalCharges  MtoM  Paperless  Check  Phone  Tenure  Dependents  \
0       45            45    29          1      1      0       0           1

   Internet
0         0
```

Apply the model with the new data:

[ ]:
```
y_prob = dt_model.predict_proba(X_new)
print('Probability of Churn_no:', round(y_prob[0,0], 3))
print('Probability of Churn_yes', round(y_prob[0,1], 3))
y_new = dt_model.predict(X_new)
print('Predicted group membership:', y_new)
```

```
Probability of Churn_no: 0.148
Probability of Churn_yes 0.852
Predicted group membership: [1]
```

This customer has an 85.2% probability of churning, so they are labeled as Churn_yes.